멀티미디어 선반입에 적용 가능한 파일 액세스 패턴 기반의 선반입 시스템

황보준형[†] · 서대화[††]

요 약

본 논문에서는 액세스 패턴의 반복성을 이용하여 비교적 적은 메모리 공간을 사용하는 SIC(Size-Interval-Count) 선반입 시스템을 제안한다. SICPS(SIC Prefetching System)는 응용 프로그램의 액세스 패턴을 예측하여 정확한 선반입을 수행하는 지식기반의 선반입 기법을 기반으로 한다. 제안된 선반입 시스템에서는 "SIC 액세스 패턴 정보"를 이용하여 반복적인 액세스 패턴을 효율적으로 저장하고, 이를 이용하여 응용프로그램의 다음에 요청될 블록을 정확하게 예측한다. 본 논문의 선반입 시스템은 일반 파일시스템에 비해 최고 40%의 응답속도 향상을 가져오며, 기존의 지식기반 선반입 기법에 비해 뛰어난 메모리 효율성을 보여준다.

Prefetching System based on File Access Pattern Applicable to Multimedia Prefetching Scheme

Jun-Hyoung Hwangbo[†] and Dae-Wha Seo[††]

ABSTRACT

This paper presents the SIC(Size-Interval-Count) prefetching system that can record the file access patterns of applications within a relatively small space of memory based on the repetitiveness of the file access patterns. The SICPS(SIC Prefetching System) is based on knowledge-based prefetching methods which includes high correctness in predicting future accesses of applications. The proposed system then uses the recorded file access patterns, referred to as "SIC access pattern information", to correctly predict the future accesses of the applications. The proposed prefetching system improved the response time by about 40% compared to the general file system and showed remarkable memory efficiency compared to the previously knowledge-based prefetching methods

1. 서 론

멀티미디어 웹서버나 대규모 과학 계산용 응용프로그램이 요구하는 파일이나 데이터의 크기가 지속적으로 증가하고 있다. 또한, 네트워크 대역폭의 확대와 컴퓨터 시스템의 처리 능력이 증가함에 따라 파일 입출력 요구량도 따라서 증가하고 있다[1]. 그리고 멀티미디어 데이터와 같은 대용량 데이터들은 재사용성이 거의 없어 한번만 요청되어 사용되고 더 이상 요청되지 않는 데이터 블록이 대부분이다. 이로 인해서 파일 버퍼 캐쉬의 적중률이 감소하고 있다. 이러한 액세스 패턴 하에서 버퍼 캐시 적중률을 향상시키기 위해서는 파일 블록 선반입 기법의 중요성이 커지고 있다.

파일 블록 선반입은 응용프로그램이 데이터를 액세스하기 전에 응용프로그램이 요청할 것으로 예측되는 데이터 블록을 미리 디스크에서 읽어서 버퍼 캐쉬에 저장함으로써 버퍼 캐쉬의 적중률을 향상시킨다. 그리고 응용 프로그램의 읽기 요청을 디스크에서가 아닌 버퍼 캐쉬로부터 해당 데이터 블록을 읽어옴으로 해서 전체 파일 시스템의 응답속도를 향상시킬 수 있다.

하지만, 파일 블록 선반입 역시 디스크 입출력 대

† 경북대학교 전자공학과 석사과정
†† 정회원, 경북대학교 공과대학 전자전기공학부 부교수

역폭을 차지하기 때문에, 선반입한 데이터 블록이 사용되지 않는다면 디스크 입출력 대역폭을 낭비하는 것이 된다. 또한 선반입한 데이터 블록을 저장하기 위해 캐시 버퍼에서 제거된 데이터 블록에 대한 액세스가 발생한다면 이를 다시 디스크에서 읽어와야 하므로, 이중으로 디스크 입출력 대역폭을 낭비하게 될 뿐만 아니라 캐시 적중률을 저하시켜 파일 시스템의 응답속도가 떨어지게 된다. 따라서 미래에 사용될 데이터 블록에 대한 예측 정확도가 높지 않다면 선반입은 오히려 파일 시스템의 성능을 저하시키게 된다.

이러한 정확하지 못한 선반입에 의해 성능이 저하되는 것을 해결하기 위해 응용프로그램의 액세스를 예측하여 선반입을 수행하기 위한 연구가 있어 왔다[2-4]. 본 논문에서는 지능적 액세스 패턴 예측 방법의 관점에서 새로운 선반입 기법인 SIC 기법을 이용한 선반입 시스템을 제안한다. 일반적인 응용프로그램의 액세스 패턴은 이전에 나타난 액세스 패턴을 다시 나타낼 가능성이 높다는 가정 하에서, SIC기법은 이전에 나타난 액세스 패턴을 사용하여 선반입을 수행한다.

SIC기법은 변화가 심한 액세스 패턴이 아니라 액세스 패턴에 규칙성이 있는 파일에 대해 선반입을 수행한다. 이러한 규칙성이 있는 액세스는 일반적인 UNIX 파일 시스템[5]이나 과학 기술 계산을 위한 병렬 처리 시스템[6,7] 등 다수의 시스템에서 평균적으로 80% 이상의 액세스를 차지한다. 또한 일반적으로 액세스되는 웹페이지의 액세스 패턴이 일정한 패턴을 가지고 있다고 가정할 때 SIC기법은 이런 웹의 액세스 패턴을 기록하여 다음 액세스 때에는 이 정보를 바탕으로 웹 서버의 응답속도를 향상시킬 수 있는 응용에도 이용될 수 있다.

따라서, SIC기법은 일반적인 파일 입출력 요청이나 웹페이지 요청이 있을 때 규칙성이 있는 액세스 패턴에 대해서 선반입할 수 있다면 충분한 성능향상을 이룰 수 있게 된다.

SICPC는 패턴 정보 저장의 공간 효율성을 보완한 것으로 액세스 정보의 메모리 사용량을 줄일 수 있게 액세스의 반복성을 이용한다. 이 시스템은 SIC 선반입 기법을 수행하기 위해서 SIC 액세스 패턴을 수집하는 액세스 패턴 수집기와 수집된 액세스 패턴을 사용해서 선반입을 수행하는 선반입 실행기의 두 부분으로 구성된다. 그리고 이 시스템은 저장된 액세스 패턴 정보를 이용한 응용프로그램의 응답속도의 향상뿐 아니라 반복적인 액세스 패턴의 정보를 압축 저장하는 기법으로 기존 시스템에 비해 뛰어난 메모리 사용 효율을 보여준다.

본 논문은 서론에 이어 2장에서는 파일 액세스 패턴 연구 및 기존의 선반입 정책 연구에 대해 알아보고 기존 연구의 문제점을 고찰한다. 3장에서는 본 논문에서 제안하는 새로운 선반입 시스템인 SICPC에 대해 알아보고 4장에서는 구현된 선반입 시스템의 성능을 평가하기 위해 기존의 파일 액세스 패턴 연구에서 알려진 액세스 패턴을 사용해서 실험한 결과를 제시한다. 마지막으로 5장에서는 실험 결과를 바탕으로 결론을 내리고 앞으로의 과제를 제시한다.

2. 파일 선반입 기법

캐쉬사용은 파일접근의 지역성이 클 경우 입출력 횟수를 줄임으로써 파일 블록들의 빠른 접근을 가능하게 한다. 하지만 사용되는 파일의 크기가 점점 커지고 한 번만 읽혀지는 데이터 블록들의 증가로 캐쉬의 효율성 문제가 발생한다[8]. 이런 한계를 보완하기 위해 앞으로 사용될 블록들을 미리 가져오는 선반입 기법은 디스크에 대한 접근 시간을 줄이는 좋은 방법이 된다

대표적인 선반입 정책에는 OBL(One Block Lookahead)과 IBL(Infinite Blocks Lookahead)같은 알고리즘 선반입 정책과 액세스 패턴을 이용하는 지식기반의 선반입으로 구분된다[9]. 이 선반입 기법들에 대해 알아보면 다음과 같다.

OBL은 가장 대표적인 선반입 정책으로서 OBA(One Block Ahead)라고 불리기도 한다. OBL 선반입 정책은 현재 읽기가 요청된 블록의 다음 블록을 선반입하는 방법으로 파일의 블록을 차례대로 액세스하는 순차적인 액세스가 많은 경우에 효과적이다.

IBL은 OBL을 확장한 선반입 정책으로 읽기가 요청된 블록의 다음 블록부터 순차적으로 디스크버퍼가 허락하는 한도 내에서 최대한 선반입을 수행한다. IBL은 순차적인 액세스가 대부분인 경우, 그리고 입출력이 많은 응용프로그램의 경우에 효과적이다. 즉, IBL은 입출력 요구가 많은 상황에서는 한번에 선반입하는 양을 늘임으로써 저장장치의 지연시간에 의한 오버헤드를 줄일 수 있어 OBL보다 높은 성능을

낸다.

이들 선반입 기법에서는 대부분의 액세스가 순차적 액세스 패턴이거나 simple-strided 액세스 패턴에서는 충분한 성능 향상을 기대할 수 있다[10].

하지만, 최근 대용량 데이터를 요구하는 멀티미디어나 웹서버 같은 응용 프로그램의 액세스 패턴이 이전에 비해 점차 복잡해지고 있어서 OBL이나 IBL의 적용이 어려워지고 있다. 따라서 최근에는 응용프로그램의 액세스 패턴을 사용하여 선반입을 수행하는 지식기반의 선반입 기법으로써 선반입의 정확성을 향상시키고 선반입을 적용할 수 있는 액세스 패턴의 범위를 확대하기 위한 연구가 활발하게 진행되고 있다.

응용프로그램의 액세스 패턴을 사용해서 선반입의 정확성을 향상시키는 방법에는 크게 두 가지 방법이 있다. 하나는 응용프로그램이 앞으로 사용하게 될 블록에 대한 정보를 파일 시스템에 제공함으로써 파일 시스템이 이 정보를 사용하여 선반입을 수행하는 'Hint-based 선반입 기법'이고, 또 다른 하나는 파일 시스템에서 응용프로그램의 액세스 패턴을 분석하여 다음에 액세스할 블록을 예측하고 이에 따라 선반입을 수행하는 '지능적 액세스 패턴 예측 방법'이다[8,11].

Hint-based 선반입 기법은 '응용프로그램의 앞으로의 액세스를 가장 잘 알고 있는 것은 응용프로그램 자신이다.' 라는 가정 하에서 응용프로그램이 앞으로 액세스하게 될 블럭에 대한 정보를 Hint라고 하는 규정된 형식의 데이터로 작성하여 특별한 API를 사용해서 파일 시스템에 제공한다. 파일 시스템은 제공된 Hint를 바탕으로 앞으로 사용하게 될 블럭을 최소의 오버헤드로 선반입을 수행할 수 있는 방법으로 선반입을 수행하게 된다.

향상된 선반입을 사용하기 위한 Hint-based 선반입 기법의 단점은 Hint-based 선반입 기법을 위한 응용프로그램의 재작성 같은 시스템 외적인 오버헤드가 크고, 기존의 표준 입출력 API를 사용해서 작성된 응용프로그램의 경우에는 성능을 향상시킬 수 없다는 것이다. 이러한 Hint-based 선반입 기법의 단점을 해결하기 위해 파일 시스템이 응용프로그램의 액세스 패턴을 분석하여 미래의 액세스를 예측하고 선반입을 수행할 수 있는 지능적 액세스 패턴 예측 방법에 대한 연구가 있어 왔다.

기존의 지능적 액세스 패턴예측 방법에서는 액세스 패턴이 가지는 반복성에 상관없이 액세스 패턴 정보를 기록하였기 때문에 액세스 패턴 정보의 길이가 파일의 길이에 비례해서 커지게 된다. 따라서 점점 파일의 크기가 커지고 있는 현재의 상황에서는 액세스 패턴정보가 차지하는 저장장치 사용량이 증가하게 된다. 이러한 문제점을 해결하기 위해서는 액세스 패턴 정보의 크기를 줄일 수 있는 방법이 필요하다.

다음 장에서는 이런 문제를 해결하기 위해 파일의 액세스가 반복적인 패턴일 때는 액세스 패턴의 정보를 압축하여 저장하는 Size-Interval-Count 액세스 패턴 기반의 선반입 시스템에 대해 알아본다.

3. SIC 선반입 시스템

이 장에서는 SIC 선반입 기법을 수행하기 위한 SICPS의 구조와 SICPS를 구성하는 액세스 패턴 수집기와 선반입 실행기의 구조 및 세부 모듈의 작동에 대해 설명한다.

3.1 SIC 선반입 시스템 구조

SICPS는 SIC 선반입 기법을 수행하기 위해서 파일 액세스 패턴을 수집하여 SIC 액세스 패턴 정보로 저장하는 액세스 패턴 수집기와 수집된 액세스 패턴을 사용해서 선반입을 수행하는 선반입 실행기의 두 부분으로 구성된다.

액세스 패턴 수집기는 응용프로그램의 읽기 액세스를 수집해서 SIC 액세스 패턴 정보의 형태로 작성해서 파일로 저장한다. 액세스 패턴 정보를 메모리에 저장하지 않고 파일로 저장하는 것은 시스템 리셋 등에 의해 액세스 패턴 정보가 소실되는 것을 방지하기 위해서이다.

SIC 선반입 기법은 지식 기반 선반입 방법의 일종이기 때문에 액세스 패턴 정보가 축적될수록 선반입의 정확성이 향상되게 된다. 만약, 시스템을 리셋할 때마다 액세스 패턴 정보를 새로 수집해야한다면 시스템 가동 초기에는 선반입의 성능이 저하되거나 선반입을 수행할 수 없게 되어 시스템의 파일 입출력 성능이 저하되게 된다.

선반입 실행기는 파일로 저장된 액세스 패턴 정보를 메모리로 읽어들여 선반입을 수행하게 된다. 파일

로 저장된 액세스 패턴 정보가 여러 개 존재한다면 선반입 실행기는 모든 액세스 패턴 정보를 읽어들여서 응용프로그램의 액세스와 비교해서 일치하는 액세스 패턴을 제외한 나머지 액세스 패턴을 메모리에서 제거하게 된다.

그림 1은 SICPS의 구조를 보여주고 있다.

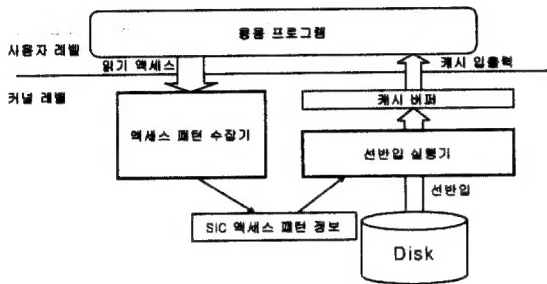



그림 1. SICPS(SIC Prefetching System)의 구조

3.2 SIC 기반의 선반입 기법

SICPS는 지식 기반 선반입 방법을 기반으로 한다. 즉, 어떤 파일을 open해서 close할 때까지의 읽기 액세스를 시간 순서대로 기록해서 저장해 두고, 이후 파일 액세스 시에는 과거에 기록된 액세스 순서 정보와 현재의 액세스 패턴을 비교해서 다음에 액세스할 데이터 블록을 예측하는 방법이다. 이러한 방법의 문제점이 패턴정보 저장의 공간 효율성을 보완하기 위해 SICPS에서는 액세스 패턴 정보의 메모리 사용량을 줄이는 액세스의 반복성을 사용하였다.

액세스의 반복성이란 그림 2와 같이 파일을 액세스 할 때 전체 파일을 연속적으로 액세스하는 것이 아니라 일정량의 데이터를 읽고 나서 일정 부분을 뛰어넘고 다시 일정량의 데이터를 읽어들이는 액세스 패턴이 반복적으로 발생하는 것을 의미한다.

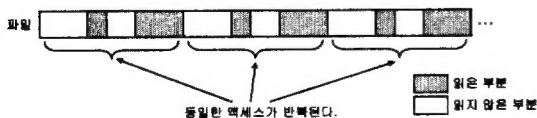



그림 2. 액세스의 반복

SICPS에서는 파일에 대한 액세스를 기록할 때, 전체 액세스를 기록하지 않고 반복단위를 구성하는 읽은 부분의 크기(size)와 읽은 부분과 다음 읽은 부분 사이의 간격(interval), 그리고 이 반복단위의 반복회수(count)를 기록함으로써 전체 액세스를 기록하는 것에 비해 액세스 패턴 정보의 크기를 줄이고 있다.

선반입 수행 시에는 이와 같이 기록된 SIC 액세스 패턴 정보를 응용프로그램의 액세스와 비교해서 다음에 발생할 액세스를 예측한다. 과거에 발생한 액세스 패턴 정보와 현재 응용프로그램이 액세스하는 것을 비교하여 현재 요청된 액세스에 뒤이어서 발생할 액세스를 과거에 발생한 액세스와 동일할 것이라고 가정하고 이 액세스들이 응용프로그램에서 요청하기 전에 선반입하여 캐시 버퍼에 저장하는 것이다.

응용프로그램의 액세스가 과거와 동일하다면 이러한 예측방법은 캐시 적중률을 100%에 가깝게 향상시킬 수 있고, 비슷하더라도 어느 정도의 성능 향상이 있게 된다. 또한, 과거의 액세스와 현재의 액세스가 다르다면 이러한 예측방법은 선반입의 정확성을 저하시키게 되므로, 계속 현재의 액세스와 과거의 액세스를 비교하여 일치하는지 확인하고, 일치하지 않는다면 선반입을 중지시킴으로써 파일 입출력 성능이 저하되는 것을 막는다.

3.3 SIC 액세스 패턴 수집기

3.3.1 SIC 액세스 패턴 수집 기법

파일 액세스 패턴 수집이란 응용프로그램의 액세스를 발생 순서에 따라 기록하여 선반입을 수행하기 위해 필요한 액세스 패턴 정보의 형태로 만드는 작업을 말한다.

SIC 액세스 패턴 정보는 패턴 테이블(PT, Pattern Table)의 패턴 테이블 엔트리(PTE, Pattern Table Entry) 형태로 기록되게 되는데 PT안의 PTE 순서에 따라 시간적인 순서를 나타낸다. 즉, 앞부분에 기록된 PTE가 뒷부분에 기록된 PTE에 비해 시간적으로 앞서 발생한 것이다.

그림 3에서 이용된 S_t는 t시간에서의 읽기 요청 크기를 나타내며, I_t는 t-1시간에서의 읽기 요청이 수행 종료되었을 때의 오프셋과 t시간에서의 읽기 요청이 시작될 때의 오프셋간의 거리, 즉 읽기 요청간의 거리를 나타낸다. S_p_i는 i번째 액세스 패턴의 읽기 크기를 나타내고, I_p_i와 C_p_i는 각각 i번째의 액세스 패턴의 읽기 액세스간의 거리와 액세스 패턴의 반복회수를 의미한다.

이와 같은 하나의 패턴이 파일의 처음부터 끝까지 반복해서 발생한다면 SIC 액세스 패턴 정보는 하나

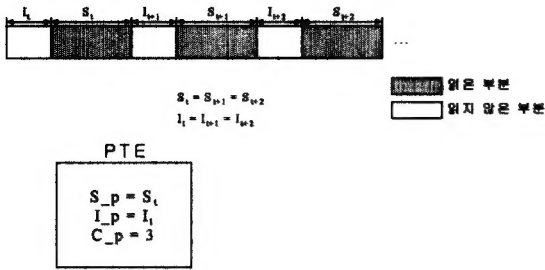



그림 3. 실제 액세스 인자와 SIC 액세스 패턴 정보 인자의 관계

그림 4. 액세스 패턴 수집기의 구조

의 PTE로 모든 액세스를 나타내게된다.

SIC액세스 패턴 수집의 방법은 크게 단순 반복 액세스 패턴, 이중 반복 액세스 패턴의 두가지 형태로 나타난다. 이중 반복 액세스 보다 더 복잡한 형태의 액세스 패턴 즉, 반복성을 가지지 않는 패턴과 같은 형태의 액세스 패턴을 가지는 응용프로그램의 존재는 그리 많지 않다. 또한 삼중, 사중 반복 액세스 패턴을 압축해서 기록하는 것은 오히려 오버헤드가 되어 액세스 패턴 수집의 성능을 저하시킨다.

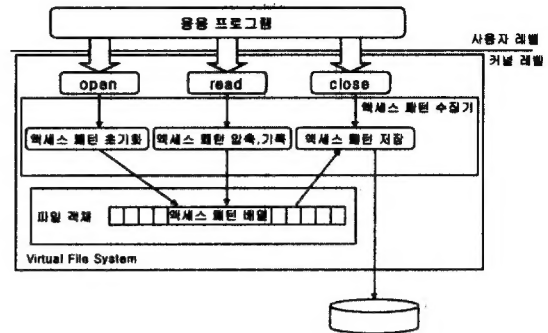
3.3.2 액세스 패턴 수집기

액세스 패턴 수집기는 응용프로그램의 읽기 액세스를 관찰하여 SIC 액세스 패턴 정보의 형태로 수집하고 디스크에 파일로 저장하는 역할을 수행한다. 응용프로그램의 읽기 액세스를 관찰하기 위해서는 파일 시스템의 인터페이스 부분에 위치하게 된다.

그림 4는 액세스 패턴 수집기의 세부 구조를 보여준다. 액세스 패턴 수집기는 액세스 패턴 초기화 모듈, 액세스 패턴 압축 기록 모듈, 액세스 패턴 저장 모듈의 주요 모듈로 구성되어 있고 부가적으로 파일 객체[1)] 내에 액세스 패턴 정보를 수집하기 위한 액세스 패턴 배열을 가지고 있다.

액세스 패턴 수집기의 각 모듈들 사이에는 상호작용이 없도록 설계되었으며 각 모듈들은 오직 응용프로그램의 파일 open, read, close 요청에 연계되어 동작한다.

(1) 액세스 패턴 초기화 모듈

액세스 패턴 초기화 모듈은 응용프로그램이 파일 open을 호출하면 파일 시스템 내부에 생성되는 파일 객체에 액세스 패턴을 기록할 수 있는 액세스 패턴 배열을 생성하고 초기화한다.

SIC 선반입 기법을 적용하기 위한 대상은 파일을 여러 번에 걸쳐서 액세스할 정도로 큰 파일이다. 따라서, 파일의 크기가 정해진 최소 크기보다 커야만 액세스 패턴 초기화 모듈에서 액세스 패턴 배열을 초기화한다. 만약 기준 이하의 크기를 가진 파일이라면 액세스 패턴 초기화 모듈은 액세스 패턴 배열을 초기화하지 않고 이를 액세스 패턴 압축 기록 모듈과 액세스 패턴 저장 모듈에 알린다.

정해진 최소 크기보다 큰 파일이라면 액세스 패턴 초기화 모듈은 액세스 패턴 배열을 생성한다. 이때, 생성하는 액세스 패턴 배열의 길이는 정해진 최대 액세스 패턴 길이이다.

(2) 액세스 패턴 압축, 기록 모듈

응용 프로그램이 파일 시스템에 읽기 요청을 하면 액세스 패턴 압축, 기록 모듈은 액세스 패턴 배열에 시간 순서대로 기록하게 된다. 이때, 액세스 패턴 정보의 길이를 줄이기 위해서 3장에서 언급한 SIC 액세스 패턴 수집방법을 사용하여 액세스 패턴 정보를 압축하여 기록한다.

기록하는 액세스 패턴의 길이가 최대 제한 배열 개수를 넘게 되면 이러한 액세스 패턴은 불규칙한 액세스 패턴으로 판단하여 더 이상 액세스 패턴 수집을 수행하지 않는다. 또한 불규칙한 액세스 패턴으로 판단되었을 경우에는 수집된 액세스 패턴 정보를 저장하지도 않는다. 따라서 이러한 액세스 패턴에 대해서는 SIC 선반입 기법을 적용하지 않게 된다. 그림 5는 액세스 패턴 압축 기록 모듈의 동작과정을 순서도로 나타낸 것이다.

(3) 액세스 패턴 저장 모듈

액세스 패턴 저장 모듈은 응용 프로그램이 파일에

1) 응용프로그램이 파일을 open하면 생성되고 close하면 소멸되는 커널 내의 객체. 파일의 ID, 오프셋, 파일의 속성 등의 정보를 가진다.

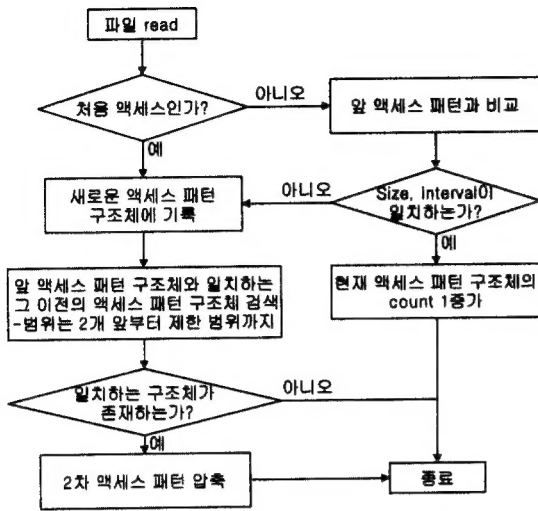



그림 5. 액세스 패턴 압축 기록 모듈의 동작

대한 close를 요청하면 수집된 액세스 패턴을 디스크에 저장한다. 그림 6은 액세스 패턴 저장 모듈의 동작 과정을 순서도로 나타낸 것이다.

다양한 액세스 패턴을 가지는 파일에 대해서는 여러 개의 액세스 패턴 정보를 유지하여 선반입 실행기에서 이를 사용하여 선반입을 실행할 수 있도록 하는 것이 선반입의 성능을 향상시킬 수 있지만, 너무 많은 액세스 패턴 정보를 저장해 두는 것은 디스크의 저장공간을 많이 사용하게 되고, 선반입 실행기에서 사용할 때도 많은 메모리를 사용하기 때문에 오버헤드가 커진다. 또한, 여러 개의 액세스 패턴 정보 중에서 자주 나타나는 것은 소수이고 나머지는 매우 드물게 발생한다면 드물게 발생하는 액세스 패턴 정보는 제거하는 것이 디스크 공간 절약 면에서 이점을 가진다.

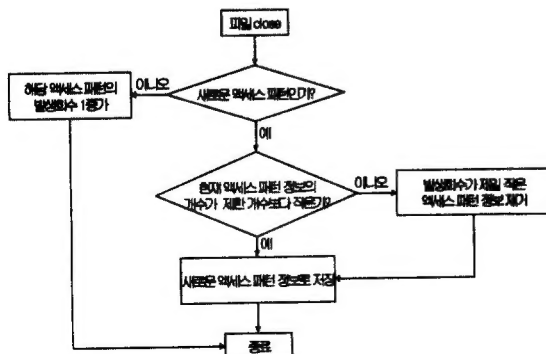



그림 6. 액세스 패턴 저장 모듈의 동작

따라서, 저장할 수 있는 액세스 패턴 정보의 개수를 제한해야만 하는데, 이렇게 되면 액세스 패턴 정보간의 교체 정책이 필요하다. 즉 새로운 액세스 패턴을 기록하기 위해서 이전에 저장된 액세스 패턴 정보 중에서 앞으로 나타날 확률이 가장 작은 패턴을 선별하여 제거할 수 있는 방법이 필요하다는 것이다. 본 논문에서는 이러한 교체정책으로 LFU정책을 사용하였다. LFU정책은 각 액세스 패턴의 발생회수를 기록하고 그 중에서 가장 작은 발생회수를 가지는 액세스 패턴을 제거하고 새로운 액세스 패턴을 기록하는 방법을 말한다.

그림 7은 여러 개의 액세스 패턴 정보를 디스크에 저장할 때의 파일들의 계층구조를 보여준다. Configuration 파일은 각 액세스 패턴 정보의 파일 이름과 각 액세스 패턴 정보의 발생회수에 대한 정보를 가지고 있다. 또한, 최대 허용 액세스 패턴 정보의 수도 포함하고 있다. 현재 저장된 액세스 패턴 정보의 수가 최대 허용 액세스 패턴 정보의 개수와 같다면 새로운 액세스 패턴 정보를 저장하기 위해서는 액세스 패턴 정보 중에서 가장 발생회수가 작은 액세스 패턴 정보를 제거하고 새로운 액세스 패턴 정보를 기록한다.

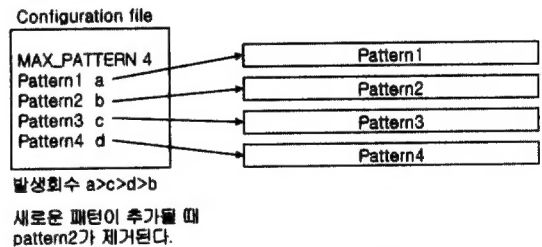



그림 7. 액세스 패턴 정보의 저장 구조

3.4 선반입 실행기

선반입 실행기는 응용프로그램이 파일을 open하면 이전에 저장된 액세스 패턴 정보를 load하여 이 정보를 사용해서 선반입을 수행한다. 선반입을 수행하기 위해서는 선반입의 대상과 시기, 선반입할 데이터 양을 결정할 수 있는 방법이 필요하다. 이러한 방법은 앞에서 언급한 SIC 선반입 기법에 의해 결정되고 선반입 실행기는 이에 맞추어 선반입을 수행한다.

그림 8은 선반입 실행기의 세부 구조와 선반입 실행기의 주요 모듈인 액세스 패턴 load 모듈, 액세스 패턴 비교 모듈, SIC 선반입 실행 모듈, OBA 선반입

실행 모듈을 보여주고 있다.

각 모듈들은 응용프로그램의 파일 open, read 요청과 연계되어 동작한다.

응용 프로그램이 파일 open을 요청하면 액세스 패턴 load 모듈은 저장된 액세스 패턴 정보를 모두 읽어들여서 파일 객체의 내부에 저장한다. 파일 read를 요청하면 SIC 선반입 실행기는 SIC 선반입 기법에 따라 선반입을 실시한다. 액세스 패턴 비교 모듈은 여러 액세스 패턴을 가지는 파일의 경우에 load된 액세스 패턴과 현재의 응용프로그램의 액세스를 비교하여 일치하지 않는 액세스 패턴 정보를 선별하여 메모리에서 제거한다. OBA 선반입 실행기 모듈은 액세스 패턴 정보가 존재하지 않거나 이전에 기록된 액세스 패턴 정보와 다른 액세스 패턴이 관찰되어 SIC 선반입을 실행할 수 없는 경우에 선반입을 실행한다. 파일 close가 요청되면 파일 객체가 메모리에서 삭제되면서 SIC 선반입 실행 모듈과 액세스 패턴 비교 모듈, 그리고 OBA 선반입 모듈은 그 파일에 대한 수행을 중지한다.

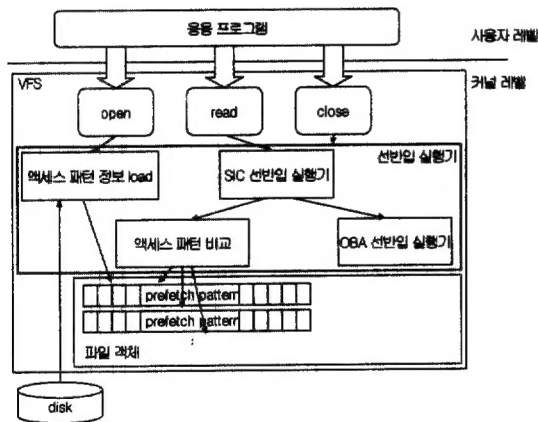



그림 8. 선반입 실행기의 구조

3.4.1 액세스 패턴 load 모듈

응용프로그램이 파일 open을 요청하면 저장되어 있는 액세스 패턴 정보를 읽어들여 파일 객체 내부에 저장한다. 또한 액세스 패턴 정보에 기록된 발생회수 역시 읽어들인다.

액세스 패턴 발생회수는 선반입에 사용할 액세스 패턴 정보가 다수 존재하고 아직 어느 액세스 패턴 정보가 현재의 액세스와 일치하는지 알 수 없을 때 선반입에 사용할 액세스 패턴 정보를 선택하기 위한 기준이 된다. SIC 선반입 기법에서는 이러한 액세스 패턴 정보중에서 가장 발생회수가 많은 액세스 패턴 정보가 현재의 액세스와 일치할 가능성이 높다고 보고 이 액세스 패턴 정보를 사용해서 선반입을 수행한다.

3.4.2 액세스 패턴 비교 모듈

여러 개의 액세스 패턴 정보를 파일 객체에 읽어들였을 때, 이 중에서 현재의 응용프로그램의 액세스 패턴과 일치하는 액세스 패턴정보를 선택할 필요가 있다. 액세스 패턴 비교 모듈은 응용 프로그램의 읽기 요청이 있을 때마다 파일 객체의 액세스 패턴 정보와 비교하여 일치하지 않는 액세스 패턴 정보를 파일 객체에서 제거한다. 따라서 현재까지의 응용프로그램의 읽기 요청과 일치하는 액세스 패턴 정보만이 파일 객체 내에 남아 있게 된다. SIC 선반입 실행기 모듈은 남아 있는 액세스 패턴 정보중에서 가장 발생회수가 많은 액세스 패턴 정보를 사용해서 선반입을 수행하게 된다.

모든 액세스 패턴 정보가 파일 객체에서 제거되었다면 이것은 현재의 응용프로그램의 액세스가 새로운 액세스 패턴이라는 뜻이므로 액세스 패턴 비교 모듈은 이것을 액세스 패턴 수집기에 알려서 액세스 패턴 수집기가 응용프로그램이 파일을 close하면 새로운 액세스 패턴 정보로 디스크에 저장하도록 한다. 그림 9는 액세스 패턴 비교 모듈의 동작과정을 순서도로 나타낸 것이다.

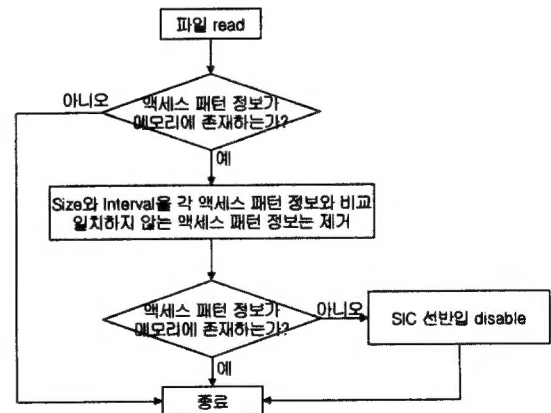



그림 9. 액세스 패턴 비교 모듈의 동작

3.4.3 SIC 선반입 실행기 모듈

선반입 실행기에서 가장 중요한 부분이다. SIC 선

반입 실행기 모듈은 파일 객체에 존재하는 액세스 패턴 정보 중에서 가장 발생회수가 많은 액세스 패턴 정보를 사용하여 선반입을 수행한다. 이 때, 사용하는 선반입 기법은 앞에서 언급한 SIC 선반입 기법을 사용한다.

파일 객체에 액세스 패턴 정보를 액세스 패턴 비교 모듈에서 모두 제거했거나 이전에 기록된 액세스 패턴 정보가 없다면 SIC 선반입 기법은 적용할 수 없다. 이러한 경우에는 OBA 선반입 실행기가 선반입을 수행하게 된다.

응용프로그램이 파일을 close할 때까지 파일 객체에 액세스 패턴 정보가 남아있다면 이것은 응용프로그램의 액세스가 액세스 패턴 정보와 일치한다는 뜻이므로 액세스 패턴 정보의 발생회수를 1증가시킨다. 반대로 파일 객체에 액세스 패턴정보가 남아있지 않다면 이것은 응용프로그램의 액세스가 현재까지 나타나지 않은 새로운 액세스 패턴이라는 뜻이므로 현재 저장된 액세스 패턴 정보의 수가 최대 허용 액세스 패턴 정보수와 같다면 가장 발생회수가 작은 액세스 패턴 정보를 제거하고 새로운 액세스 패턴 정보를 기록하고, 최대 허용 액세스 패턴 정보 수보다 작다면 새로운 액세스 패턴 정보를 저장하고 Configuration파일에 등록한다.

4. 실험 및 성능평가

이 장에서는 리눅스 VFS 상에 구현된 SICPS의 성능을 실험한 결과를 제시한다.

4.1. 실험 환경

SICPS는 리눅스 운영체제의 커널 2.2.17을 수정하여 구현하였다.

사용된 시스템의 사양은 다음과 같다.

CPU 펜티엄프로 200MHz
RAM 32MB EDO
Storage 삼성 2.1GB EIDE Hard disk

성능 실험을 위해 응용프로그램의 액세스 패턴을 다음과 같이 선정하였다.

패턴 1 : 순차적, 읽기, 100MB
패턴 2 : simple-strided, 읽기, 100MB,
패턴 3 : random, 읽기, 100MB, 1-50KB read, random lseek

패턴 1은 일반적인 시스템의 파일 액세스의 대부분을 차지하는 순차적 읽기를 사용하여 일반적인 액세스 패턴에서의 성능을 비교하기 위한 것이다. 패턴 2는 병렬 처리 과학 기술 계산 응용에서 자주 나타나는 패턴으로 일반적인 OBL나 IBL에서는 선반입을 수행할 수 없는 패턴으로서 새로운 선반입 정책이 어느 정도의 성능 향상을 보이는 지를 알아보기 위한 것이다. 패턴 3은 패턴을 기록할 수 없는 패턴을 사용하여 제안된 선반입 정책이 이러한 패턴에 대해서는 성능이 좋지 않음을 보이기 위한 것이다.

4.2 SIC 선반입 인자

최적의 성능을 내는 선반입 양과 선반입 윈도우의 크기는 파일 입출력 시스템의 성능과 캐시 버퍼의 크기 등의 실제 시스템의 성능에 의해 결정된다.

본 논문에서는 실험적으로 최적의 성능이 얻어지는 선반입 양을 구했다. 실험에 의해 결정된 최적의 선반입 양은 읽기 크기(Size)가 1 page(=4KB) 이하일 때는 2 page(=8KB) 이고, 읽기 크기가 1 page보다 클 때는 읽기 크기의 2배이다. 이보다 적은 양을 선반입하게 되면 캐시 적중률이 저하되게 되고, 이보다 많이 선반입하게 되면 캐시 적중률은 거의 같지만, 처음 액세스가 시작될 때, 선반입에 걸리는 시간이 길어져서 첫 번째 읽기 요청에 대한 응답시간이 길어진다.

본 논문에서 실험적으로 알아낸 최적의 선반입 시기가 되는 선반입 윈도우의 크기는 선반입 양의 3배이다. 이보다 더 작은 크기에서는 캐시 적중률이 저하되고 응용프로그램의 수행속도가 떨어졌고, 이보다 더 큰 크기에서는 캐시 적중률은 거의 일정하지만 응용프로그램의 첫 번째 읽기 요청 시에 선반입하는 양이 많아져서 응답속도가 떨어졌다.

4.3 실험 결과

SIC 선반입 기법은 이전에 저장된 액세스 패턴 정보와 동일한 액세스 패턴에 대해서만 선반입을 수행할 수 있다. 실험에서는 각 액세스 패턴에 대해 한 번의 실행을 거쳐서 액세스 패턴 정보를 수집할 수 있도록 하고 실험을 실시하였다.

선정한 각 액세스 패턴에 대해 5회의 측정을 실시하였고 이 중에서 최대와 최소를 제거한 나머지 3회

의 측정값을 평균해서 결과를 도출하였다. 비교대상이 되는 정책은 리눅스의 OBA 정책으로서 OBL 정책의 변형이다. 리눅스 OBA 정책은 순차적 읽기에 최적화되어 있고 순차적 읽기가 계속되면 선반입하는 크기를 늘리며, 정해진 크기 이상의 lseek가 발생하면 선반입을 중지한다. 이후에 순차적 읽기가 감지되면 다시 선반입하는 크기를 조금씩 늘려간다.

4.3.1 순차적 액세스 패턴

순차적 읽기 액세스 패턴에 대해 리눅스의 선반입과 구현된 선반입 정책을 적용하여 실험을 실시하였다. 테스트 프로그램은 한번에 1KB-64KB 크기를 파일의 처음부터 끝까지 순차적으로 읽었다.

표 1은 순차적 읽기 액세스 패턴에 대한 수행 시간을 읽어들이는 단위를 1KB에서 64KB까지 변화시키면서 측정한 결과이다. 그림 10은 표 1의 결과를 그래프로 나타낸 것이다.

표 2는 이 실험에서 측정된 캐시 적중률로서 선반입의 정확성을 보여주고 있다.

그림 10에서 볼 수 있듯이 4KB 이하의 읽기 크기

표 1. 순차적 읽기 수행 시간 결과(단위 : 초)

	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	51.02	25.06	12.89	12.11	10.49	10.53	11.01
SIC scheme	51.93	25.07	12.36	8.74	8.81	8.81	8.81

표 2. 순차적 읽기 수행 캐시 적중률(단위 : %)

	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	98.74	97.90	96.86	96.86	96.87	96.88	96.88
SIC scheme	99.99	99.99	99.99	99.98	99.98	99.96	99.99

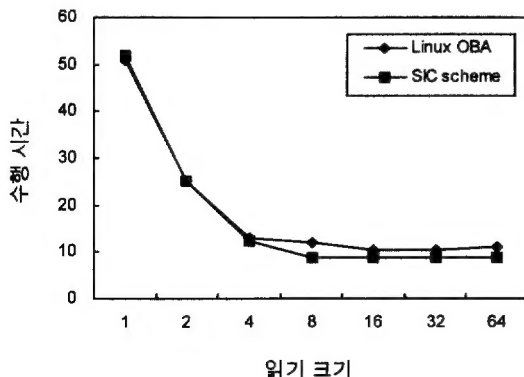



그림 10. 순차적 읽기 수행 시간

에서는 거의 같은 성능을 내지만 그 이상의 크기에대해서는 최소 16%에서 최대 27%의 성능 향상이 있음을 확인할 수 있다.

또한, 수행 시간이 1KB에서 4KB 까지는 급격히 감소하다가 4KB 이후에는 거의 일정한 것을 볼 수 있는데, 이것은 리눅스에서 사용하는 최소 입출력 단위가 4KB이기 때문이다. 즉, 1KB를 4번 연속 접근하는 것은 4KB 크기를 4번 접근하는 것과 같기 때문에 4배정도의 시간이 걸리게 된다. 하지만 4KB 크기를 넘는 액세스의 경우에는 이러한 중복 접근이 발생하지 않기 때문에 읽어들이는 크기가 같다면 거의 같은 시간이 걸리게 된다.

표 2에 나타난 바와 같이 리눅스의 선반입 정책은 순차적 읽기에 최적화되어 있어서 96%이상의 캐시 적중률을 유지하는 것을 알 수 있다. 또한, 제안한 SIC 정책에서는 순차적인 액세스 패턴에 대해 거의 100%에 가까운 확률로 캐시 적중률을 향상시킬 수 있음을 확인할 수 있다.

4.3.2 simple-strided 액세스 패턴

Simple strided 패턴은 과학 기술 계산을 위한 병렬 응용프로그램에서 발견되는 액세스 패턴으로서 일정한 길이의 액세스와 일정한 길이의 lseek을 반복하는 것이다. 여기에서는 lseek 거리를 고정하고 읽기의 크기를 변화시켜서 그 결과를 관찰하였다.

표3은 lseek거리를 4096byte로 고정하고 읽기 크기를 1KB에서 64KB까지 변화시키면서 그 읽기에 걸린 수행시간을 기록한 것이다. 표4는 캐쉬 적중률을 기록한 것이고 그림 11은 수행시간을 그래프로 나타낸 것이다.

결과에서 알 수 있듯이 lseek 간격이 크기 때문에 리눅스의 OBA는 수행시간과 캐시 적중률 모두에서 성능이 저하되었다. 하지만, SIC 정책에서는 상대적 매우 높은 확률로 다음 블럭을 선반입함을 알 수 있다. 리눅스의 OBA에서는 캐시 적중률이 최저 33.46%까지 하락하였지만 SIC 정책에서는 대부분의 읽기 크기에서 99% 이상의 캐시 적중률을 보이고 있다. 또한, 읽기 수행 시간에서도 최소17%에서 최대 43%까지 성능이 향상되었다.

여기에서도 읽기 크기가 커짐에 따라 순차적 읽기에 가까워 져서 리눅스의 OBA 선반입 정책의 캐시 적중률이 상승하는 것을 볼 수 있다.

표 3. 4096 byte strided에서의 읽기 수행 시간(단위 : 초)

단위 : 초	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	15.77	15.66	15.88	15.62	14.24	12.64	11.36
SIC scheme	9.78	10.24	9.37	8.93	9.51	9.09	9.45

표 4. 4096 byte strided 패턴에서의 캐시 적중률 (단위 : %)

단위 : %	1KB	2KB	4KB	8KB	16KB	32KB	64KB
Linux OBA	77.49	41.71	33.46	66.71	83.39	91.68	96.53
SIC scheme	99.63	98.46	99.09	99.70	99.21	99.82	99.79

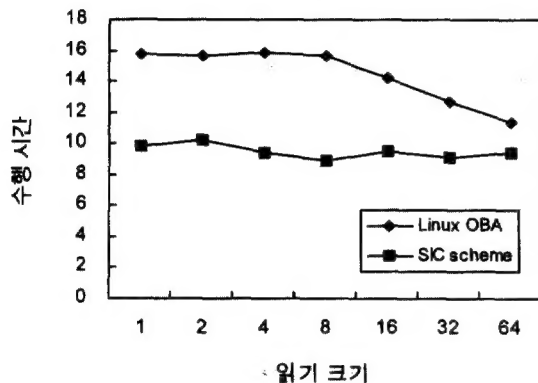



그림 11. 4096 strided 패턴에서의 읽기 수행 시간

4.3.3 2-D strided 액세스 패턴

2-D strided 패턴은 과학 기술 계산을 위한 대용량 병렬 프로그램에서 발견되는 것으로 simple strided 패턴에 비해서는 발견되는 확률이 적다. 이 실험에서는 1-D simple strided 패턴보다 복잡한 패턴을 선반입할 수 있는지 검증하기 위하여 사용되었다. 실험에 사용된 패턴은 4KB의 lseek과 16KB의 읽기를 50회 반복하고 8KB의 lseek과 8KB의 읽기를 50회 반복하는 것을 교대로 나타나도록 설정하였다.

표 5와 표 6은 각각 수행 시간과 캐시 적중률을 기록한 것이다.

이 실험에서도 SIC 정책은 리눅스 OBA에 비해 34% 정도의 성능 향상을 보여준다. 또한, 캐시 적중률에 있어서도 리눅스 OBA가 72%에 그친 반면에 99%에 가까운 성능을 보이고 있다.

5. 결론 및 향후 과제

현재의 파일 액세스 패턴은 멀티미디어 파일의 증

표 5. 2-D strided 패턴에서의 읽기 수행 시간(단위 : 초)

	1회	2회	3회	4회	5회	평균
Linux OBA	15.74	14.97	15.24	14.97	15.03	15.19
SIC scheme	9.74	9.66	10.24	9.92	10.33	9.98

표 6. 2-D strided 패턴에서의 캐시 적중률(단위 : %)

	1회	2회	3회	4회	5회	평균
Linux OBA	72.46	72.40	72.40	72.40	72.40	72.41
SIC scheme	99.33	99.27	98.52	99.12	98.66	98.98

가와 네트워크 대역폭의 확대에 따라 대용량의 파일이 증가하고 재사용의 빈도는 감소하고 있다. 이러한 상황에서는 캐시 적중률이 감소하게 된다. 캐시 적중률을 향상시키기 위해서 선반입의 필요성이 높아지고 있지만, 잘못된 선반입은 파일 입출력의 성능을 저하시키게 된다.

따라서 정확한 선반입을 통해 파일 입출력의 성능을 향상시키기 위한 목적으로 응용프로그램의 액세스 패턴에 따라 동적으로 선반입을 수행하는 지식 기반 선반입 방법이 고안되었다. 하지만, 기존의 지식 기반 선반입 방법은 응용프로그램의 액세스 패턴 정보를 저장하는데 필요한 메모리 공간이 파일의 길이에 비례해서 커지는 문제점이 있다.

본 논문에서 제안하는 SIC 선반입 시스템은 이전에 수행된 파일 액세스 패턴에 관한 연구에 의해 알려진 파일 액세스 패턴의 반복성을 기반으로 하여 액세스 패턴 정보를 압축해서 저장함으로써 지식 기반 선반입 방법의 성능을 그대로 유지하면서 메모리 사용량을 줄였고 수집된 액세스 패턴 정보를 디스크에 파일로 저장함으로써 지식 기반 선반입 방법의 문제점인 시스템 리셋 등에 의해 수집된 액세스 패턴 정보가 소실되는 문제를 개선하였다. 또한, 액세스 패턴 정보를 메모리에 캐시 해둠으로써 파일 open시에 선반입 수행을 위해 액세스 패턴 정보를 읽어들이는 오버헤드를 경감시켰다.

실험 결과에서 알 수 있듯이, 현재 가장 많이 사용되고 있는 OBA 선반입 기법이 순차적 액세스 패턴에 대해서만 충분한 성능을 보여주는데 비해서 제안하는 SIC 선반입 기법은 순차적 액세스를 포함해서 simple-strided 액세스 패턴에 대해서도 선반입을 수행해서 성능을 향상시킬 수 있다.

현재 SICPS는 과거에 나타난 액세스 패턴과 동일

한 액세스에 대해서만 선반입을 수행할 수 있다. 향후 과제로는 과거의 액세스 패턴과 완전히 동일하지는 않지만 거의 유사한 액세스에 대해서도 선반입을 적용할 수 있는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," In Proc. of IEEE Infocom '99, pp. 126-134, March 1999.

[2] T. M. Madhyastha, "Automatic Classification of Input/Output Access Patterns, Tech. Rep.," University of Illinois at Urbana-Champaign, Department of Computer Science, August 1997.

[3] T. M. Madhyastha and D. A. Reed, "Exploiting Global Input/Output Access Pattern Classification," In Proc. of SC'97, November 1997, CD-ROM.

[4] T. M. Madhyastha and D. A. Reed, "Input/Output Access Pattern Classification Using Hidden Markov Models," In Proc. of the Workshop on Input/Output in Parallel and Distributed Systems (IOPADS), November 1997.

[5] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," In Proc. of the 10th Symposium on Operating System Principles, pp. 15-24, December 1985.

[6] N. Nieuwejaar, D. Kotz, A. Purakayastha, C.S. Ellis, and M. Best, "File-access characteristics of parallel scientific workloads," IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 10, pp. 1075-1089, October 1996.

[7] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar and M. Best. "Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor," In Proc. of the Ninth International Parallel Processing Symposium, pp. 165-172, April, 1995.

[8] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. "Informed prefetching and caching". In Proc. of the Fifteenth ACM Symposium on Operating Systems Principles, pp. 79-95, December 1995.

[9] D. Kotz and C.S. Ellis. "Practical prefetching techniques for multiprocessor file systems," Journal of Distributed and Parallel Databases, vol. 1, no. 1, pp. 33-51, January 1993.

[10] T. Cartes, "Cooperative Caching and Prefetching in Parallel/Distributed File Systems," Ph.D Thesis, Universitat Politecnica de Catalunya, 1997.

[11] A. Tomkins, R.H. Patterson and G.A. Gibson, "Informed Multi-Process Prefetching and Caching," In Proc. of the ACM International Conference on Measurement and Modeling of Computer Systems(SIGMETRICS), June 1997.

황 보 준 형

2000년 경북대학교 전자전기공학부 졸업(학사)
2000년~현재 경북대학교 전자공학과 (석사과정)
관심분야 : 병렬분산처리, 클러스터 컴퓨팅, 멀티미디어 선반입 시스템
E-mail : bluesky@palgong.knu.ac.kr

서 대 화

1981년 경북대학교 전자공학과 (학사)
1983년 한국과학기술원 전산학과 (석사)
1993년 한국과학기술원 전산학과 (박사)
1981년~1995년 한국전자통신연구소 시스템 S/W 연구실 근무
1998년~1999년 University of California Irvine 연구 교수
1995년~현재 경북대학교 공과대학 전자전기공학부 부교수
관심분야 : 병렬분산처리, 운영체제, 병렬처리, 컴퓨터 구조
E-mail : dwseo@ee.knu.ac.kr